Prescriptive Design Patterns

Proactive Guidance for Real-World Systems

Kevin Mullet
REACTOR Experience Design

Patterns are Good

- Capture Real Design Insights
- Represent Knowledge in Structured Form
- Impose Discipline on Design Team
- Encourage Designers to Think Globally

Problem is the **Effort** of Creating & Using Them

Patterns (Today) are Not So Good

- Often perceived as overly constraining
- Existing collections are hopelessly incomplete
- Generally fail to proactively “lead” to a solution
- Require lots of knowledge to apply correctly

Control of **Abstraction** is the Key to Success

Background and Experience

- Software Design Education
    - *Industrial Design, Experimental Psychology, Computer Science*
    - *BS & MA from The Ohio State University*
- Industry Experience
    - *Aaron Marcus + Associates*
    - *Sun, Macromedia, Netscape*
    - *Icarian, Scoutfire, Propel*
    - *REACTOR Experience Design*

Recent Work and Current Focus

- Pattern Language for eCommerce
  - *Design Patterns*
  - *Components and Frameworks*
  - *Reference Implementations*
- Patterns for Rich Internet Applications
  - *Forward-Looking (Patterns for "New" Phenomena)*
  - *Prescriptive (Provide Concrete Guidance)*
  - *Pragmatic (Useful to "Real-World" Practitioners)*

What Practitioners Want

- Practitioners tend to be highly pragmatic
- They demand very concrete examples
- They resist what seems like needless abstraction
- They want patterns to *lead* them to good design

Existing PL’s Don’t Tell You **How** to Get There

What Practitioners Need

- Tools that capture complexity of real world
- Single integrated repository for design info
- “Patterns” for things like GUI Standards
- “Patterns” for conventional application designs

Need a **Broader** Sense of “Pattern-ness”

Prescriptive Pattern Languages

- Must guide practitioners to effective designs
- Must address more than just general solutions
- Must be extensible and specific
- Must provide a common integration framework

Must Clarify **Relationships** Between Patterns

Types of Pattern Relationships

- **Aggregation (“has-a”)**
  Pattern B is part of Pattern A
- **Derivation (“is-a”)**
  Pattern B is a specialized form of Pattern A
- **Reference (“uses”)**
  Pattern A requires the presence of Pattern B

Aggregation

- Alexandrian pattern languages are structured around aggregation hierarchies (networks)
- Very natural and concrete compositions
- Necessarily quite general

Forms the **Basis** for a Prescriptive Framework

Pattern Aggregation

- Application
  - *Main Window*
    - Navigation Framework
      - Current Path
      - Destination Links
      - History
      - Favorites
    - Content Display

Derivation

- Derivation captures the evolution from general to increasingly specialized solutions
- Also natural, but orthogonal to composition
- Confusion results when the two are mixed

Uses Inheritance to Manage **Complexity**

Pattern Derivation

- Application
  - *Desktop Application*
    - Content Editor
      - Word Processor
      - Spreadsheet
      - Illustration
      - Presentation
      - Authoring

Reference

- Reference relationships link patterns that are intended to be *used* together
- Can be used to simplify pattern hierarchies
- Can be used to filter pattern languages

Links Patterns **Together** to Create Solutions

Beyond “Related Patterns”

Name	
Type	Derivation (Parent)
Needed for	Aggregation (Parents)
Context	
Forces	
Summary	
Components	Aggregation (Children)
Connections	Reference (Links)
Rationale	
Examples	
Alternate Forms	Derivation (Peers)
Special Cases	Derivation (Children)

Example

Name	Navigation Path
Type	Navigation Link
Needed for	Navigation Framework
Context	Hierarchical Information Spaces
Forces	Need to see current label and full path
Summary	Make each path component a link, and separate them w/ a recognizable char
Components	Path Elements, Element Separator
Connections	Type Style, Link Cues, Category Names
Examples	Home > All Categories > Computer and Office Products
Alternate Forms	Path Menu
Special Cases	Abbreviated Path

Where We Stand

- Several isolated pattern “languages”
- All have some good qualities
- Little or no connection between them
- Need to leverage the Net to share work on converging and extending this work

What You Can Do

- Use prescriptive patterns as a formal structure to make your design work more generalizable
  - *Think systematically beyond the immediate need*
  - *Be application-agnostic when defining behavior*
  - *Write **two** patterns if necessary (derive one)*
- Help advance the art: Share your own patterns and provide feedback on the work of others